REMARKS

Claim 11 stands rejected under 35 U.S.C. §112, second paragraph, as being indefinite. Claim 11 has been amended in a readily apparent manner to overcome this rejection. Withdrawal of the rejection is respectfully requested.

Claims 1-12 stand rejected under 35 U.S.C. §102(a) as being anticipated by Goodnow, II et al., US 5,590,329 (Goodnow). Applicants respectfully traverse with respect to independent claims 1 and 11, because the cited reference does not disclose or suggest that the specified areas in the memory are determined by obtaining starting and ending addresses in an object file where an object code relating to the target dynamic variable is stored, as now described in claims 1 and 11.

The Examiner asserts that the area specifying means for specifying areas is disclosed by Goodnow which teaches that programmers "often allocate one large block of memory and then break the large block into smaller pieces, as needed." (Col. 20, lines 47-48).

As amended, claims 1 and 11 now more specifically describe that the mechanism for determining the specifying areas by the area specifying means, i.e., obtaining starting and ending addresses in an object file where an object code relating to the target dynamic variable is stored. This feature is not disclosed or suggested in Goodnow. For this reason, claim 1 and its dependent claims 2-10 and claim 11 are allowable over the cited reference.

Moreover, the dynamic variables in the present invention correspond, for example, to COMMON variables in FORTRAN, and as shown in Fig. 9, it is necessary to integrate the same area declared in a plurality of subroutines by a linker. When a compiler generates an object, variable areas are not allocated and there is no allocation library call. Allocation is performed by an OS. In the present invention, initial values are substituted for these dynamic variables with the initialization library 22a (see Fig. 4). As shown in Fig. 6, this involves passing information from the linker to the initialization library.

However, with variables shown in Fig. 2b of Goodnow, when a compiler finishes generating an object, variable areas have been allocated or an allocation library call is generated. Goodnow does not describe where and how the areas are initialized. Therefore, the method for initializing the areas will differ from that of the present invention as described in claim 1.

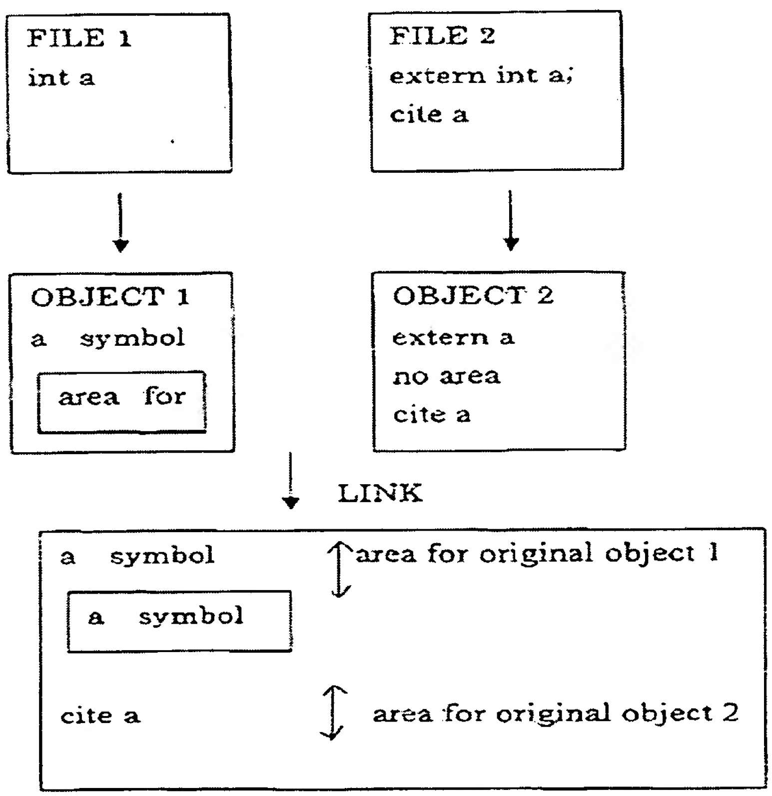
In claim 2, the term "integrate" indicates that when a compiler generates an object, the same data section is allocated to different dynamic variables declared in the same COMMON block. This is shown in Fig. 8 of the application.

In Goodnow, a large block of memory is broken into smaller pieces (see column 20, lines 55-57). Each smaller piece of allocated memory is accessible by at least one pointer. The valid range for each pointer would normally be recorded in the pointer check table as the entire large block of memory. Thus, the pointer check table should indicate the proper range for a pointer that is intended to point to a smaller piece of the larger

allocation. This differs from the contents of the term "integrate" in claim 2, and therefore does not disclose this feature.

In claim 3, the term "integrate" means that the same area declared in a plurality of subroutines is united into one by a linker (see Fig. 9 or 10). According to Goodnow, symbols are integrated (see column 6, lines 51-55). This differs from the contents of the term "integrate" in claim 3.

The following figure likely illustrates symbol integration according to Goodnow.



The resolution that "extern a" is "a" included in the object 1 is obtained.

In the present invention, as described in claim 4, a new data section is used. This data section is generated by a compiler, is integrated by a linker, and is recognized by an OS or an initialization library at execution time. This data section is an interface common to the compiler, the linker, the OS, and the initialization library. Dynamic variables, such as COMMON variables in FORTRAN, are initialized by such cooperation.

In Goodnow, the structure that is created by the wrapper function for the malloc function is described (see column 19, lines 45-47). This structure can be recognized only by the wrapper function for the malloc function and is used only by the wrapper function for the malloc function. Dynamic variables acquired by the malloc function in C can be initialized in the wrapper function for the malloc function.

In claim 5, an initializing process is performed on the new data section and is not performed on an existing data section. This ensures that even if a linker or an initialization library is generated after the present operation, an existing object will function as before.

In Goodnow, an error checking process performed on a compiled function is shown (see Fig. 12b). However, Goodnow does not mention that whether an initializing process is performed depends on the type of the data section. In particular, no mention is made of the compatible function of an existing object.

In claim 6, "initial-value entering means for entering an initial value used by the initializing means before a compiling process" means that an initial value is given in a

compile option (see the specification, page 18, line 14 to page 19, line 5). Goodnow mentions what initial value is given, but does not mention how to give an initial value.

In claim 7, the initial-value entering means for entering an initial value used by the initializing means before executing the load module, indicates that an initial value is given in an execution time option (see the specification, page 19, lines 6-14). Goodnow mentions what initial value is given, but does not mention how to give an initial value.

In the present invention, as described in claim 8, a data section on which an initializing process is to be performed can be designated in a compile option by specifying the leading character of a block name (see the specification, page 18, lines 23-26). Goodnow does not mention how to designate a variable to be initialized. The present invention is believed to be allowable for all these reasons, also.

Applicants respectfully traverse with respect to claim 12, because the cited reference does not disclose or suggest the area ensuring means for ensuring an availability of areas in a memory of a predetermined number of bytes more than areas declared in an array specified by the array specifying means.

The claimed area ensuring means is stated to be disclosed in Col. 10, line 62 of the Goodnow reference, which states that "the array dimension checking subroutine of Fig. 4 will obtain the maximum declared size for each dimension of the array from the internal symbol table 75." The reference further teaches that if "the constant reference exceeds the declared maximum value for this dimension or is a negative value, an error message will be generated in step 435" (Col. 11, lines 7-9). It would appear that Goodnow teaches obtaining

the maximum declared size of the array, and generating an error message when the declared maximum size is exceeded. This, however, does not disclose (or suggest) the claimed area ensuring means which ensures that there is an availability of areas in a memory of a predetermined number of bytes than areas declared in the array which is specified by the area specifying means. In other words, Goodnow teaches working with the actual array size declared in a dimension checking subroutine, but does not disclose (or suggest) ensuring that there is an availability of a predetermined number of bytes more than the declared size. For this reason, claim 12 is also believed to be allowable over Goodnow.

For all of the above reasons, Applicants request reconsideration and allowance of the claimed invention. The Examiner should contact Applicants' undersigned attorney if a telephone conference would expedite prosecution.

Respectfully submitted,

GREER, BURNS & CRAIN, LTD.

By B. Joe Kim

B. Joe Kim
Registration No. 41,895

February 16, 2005
Suite 2500
300 South Wacker Drive
Chicago, Illinois 60606
(312) 360-0080
Customer No. 24978